Remarks

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks.

With entry of the present amendment, claims 1-18 are pending. Claim 13 is allowed.

Patentability Over Wollrath in view of Chernick

The Office has asserted a rejection of claims 1-11 and 14-16, under 35 U.S.C. § 103(a) over U. S. Patent No. 6,253,256 ("Wollrath") in view of U. S. Patent No. 5,848,234 ("Chernick"). The Office has asserted a rejection of claim 12, under 35 U.S.C. § 103(a) over U. S. Patent No. 6,567,861 ("Kasichainula") in view of Chernick. Applicants respectfully traverse.

Claim 1

Amended claim 1 is generally directed to a system "wherein an object reference to the copy of the first method invocation recorder is passed as a parameter of the method invocation to an object of the second object class so an object of the second object class can record results for an object of the first object class."

Specifically, claim 1 recites,

> 1. An object execution system supporting passing of an object reference in a method invocation delivered via a queued message, the system comprising:
>
> an object configuration store containing object properties information representing properties of at least first and second object classes executable in the system, the object properties information designating the first and second objects as supporting queued method invocation, the second object class having a method with a parameter for passing an object reference;
>
> a method invocation recording facility operative responsive to request of a client program to supply method invocations recorders for object instances of object classes designated as supporting queued method invocations;
>
> a first method invocations recorder supplied by the method invocation recording facility at request of the client program for the first object class; and
>
> a second method invocations recorder supplied by the method invocation recording facility at request of the client program for the second object class, the second method invocations recorder operating in response to a method invocation in which an object reference to the first method invocations recorder is passed to cause a data stream representation of the first method invocations recorder to be marshaled into a method invocations message for submission into a message queue associated with the second object class;

> a method invocation play-back facility operative responsive to a queued method invocations message to supply method invocation players for object instances of object classes designated as supporting queued method invocations;
>
> a first method invocation player supplied by the method invocation play-back facility in response to the method invocations message queued to the message queue associated with the second object class, the first method invocations player operating in response to the method invocations message to unmarshal the date stream representation and create therefrom a copy of the first method invocations recorder, and to pass an object reference to the copy of the first method invocations recorder as a parameter of a method invocation to an object of the second object class;
>
> *wherein an object reference to the copy of the first method invocation recorder is passed as a* parameter *of the method invocation to an object of the second object class so an object of the second object class can record results for an object of the first object class.* (Emphasis Added).

Applicants respectfully submit that the art cited by the Office fails to teach or suggest a system "wherein an object reference to the copy of the first method invocation recorder is passed as a parameter of the method invocation to an object of the second object class so an object of the second object class can record results for an object of the first object class." These features are discussed in the application, for example, at pages 16-17.

Applicants respectfully submit that a Wollrath-Chernick combination fails to teach or suggest the recited arrangement. For example, the Office recites the following Wollrath passages,

> When receiving machine 402 receives byte stream 408, it identifies the type of transmitted object. Machine 402 contains its own RMI 410 and code 411 for processing of objects. If byte stream 408 contains a marshalled object, machine 402 may create a new object 414 using the object type identified in the marshalled object, the state information, and code for the object. Object 414 is a copy of object 405 and is stored in memory 413 of machine 402. If code 412 is not resident or available on machine 402 and the marshalled object does not contain the code, RMI 410 uses the Java URL from the marshalled object to locate the code and transfer a copy of the code to machine 402. Because the code is Java bytecodes and is therefore portable, the receiving machine can load the code into RMI 410 to reconstruct the object. Thus, machine 402 can reconstruct an object of the appropriate type even if that kind of object has not been present on the machine before. *Col. 6, ll 30-46.*
>
> ... A marshalled object is a container for an object that allows that object to be passed as a parameter in an RMI call, but preferably postpones conversion of the marshalled object at the receiving machine until an application executing on the receiving machine requests the object via a call to the marshalled object. A container is an envelope that includes the data and either the code or a reference to the code for the object, and that holds the object for transmission. The

> serializable object contained in the marshalled object is typically serialized and deserialized when requested with the same semantics as parameters passed in RMI calls. Serialization is a process of converting an in-memory representation of an object into a corresponding self-describing byte stream. Deserialization is a process of converting a self-describing byte stream into the corresponding object. *Col. 6, ll 60-67.*
>
> ... A marshalled object's constructor takes a serializable object (obj) as its single argument and holds the marshalled representation of the object in a byte stream. The marshalled representation of the object preserves the semantics of objects that are at passed in RMI calls: each class in the stream is typically annotated with either the object's code or a URL to the code so that when the object is reconstructed by a call to a "get" method, the bytecodes for each class can be located and loaded, and remote objects are replaced with their proxy stubs. The "get" method is a method called by an application to execute a process of unmarshalling, which is reconstruction of an object from a marshalled object using a self-describing byte stream (the marshalled object), and a process to obtain the necessary code for that process. A proxy stub is a reference to a remote object for use in reconstructing an object. *Col. 7, ll 34-50.*
>
> ... The machine determines if the byte stream is a marshalled object (step 602). If it is not such an object, the machine performs normal processing of the byte stream (step 603). Otherwise, if the received byte stream represents a marshalled object, the machine holds the marshalled object for later use in response to a get method invoked by a process on the receiving machine. If the receiving machine determines that the object is to be transmitted to another machine (step 604), it simply transmits the byte stream without reconstructing the object. If the machine uses the object, it performs reconstruction of the object using its RMI and associated code (step 605). If the reconstruction code for the object is not resident on the machine, it uses a URL to request and obtain the code (step 606), as described above. The machine determines if it needs to transmit the object to another machine (step 607). If the object is destined for another machine, it is transmitted as a byte stream (step 608). *Col. 8, ll 33-50.*

As understood by Applicants, the recited passages fails to teach or suggest a system "wherein an object reference to the copy of the first method invocation recorder is passed as a parameter of the method invocation to an object of the second object class so an object of the second object class can record results for an object of the first object class."

Additionally, Chernick teaches away from the recited arrangement. In Chernick, when a server is remote, an RPC mechanism is used. *See e.g., col. 13, ll 10-44.* This teaches away from claim 1. For a local server, Chernick discusses a single queue shared by client and server, *see e.g., col. 13, ll 45-67, Fig 5 and 6,* which teaches away from recited queued components. Since the

references when combined "must teach or suggest all the claim limitations," the proposed combination fails to teach or suggest amended claim 1.

For at least this reason claim 1 should be allowed. Such action is respectfully requested.

Claims 2-5

Dependent claims 2-5 recite additional patentably distinct subject matter. However, since they depend from claim 1, they are allowable for at least the reasons stated above for claim 1. Claims 2-5 should be allowable. Such action is respectfully requested.

Claim 6

Claim 6 is generally directed to a method comprising "when marshaling an interface pointer reference to the second queued component in any of the method invocations issued by the client program for the first queued component, incorporating interface passing information in the data marshaled into the message, the interface passing information designating to enqueue any method invocation by the first queued component on an interface of the second queued component referenced by the interface pointer reference into the second message queue."

Specifically, claim 6 recites,

> 6. A method of yielding results from processing work of a first queued component to a second queued component, where the work of the first queued component is initiated by method invocations delivered via a first message queue, and the second queued component is dispatched method invocations delivered into a second message queue, the method comprising:
>
> responsive to a client program issuing a first set of method invocations for the first queued component, marshaling data for the method invocations of the first set into a message to be enqueued into the first message queue; and
>
> *when marshaling an interface pointer reference to the second queued component in any of the method invocations issued by the client program for the first queued component,* incorporating *interface passing information in the data marshaled into the message, the interface passing information designating to enqueue any method invocation by the first queued component on an interface of the second queued component referenced by the interface pointer reference into the second message queue.* (Emphasis Added).

Applicants respectfully submit that the art cited by the Office fails to teach or suggest a method comprising "when marshaling an interface pointer reference to the second queued component in any of the method invocations issued by the client program for the first queued component, incorporating interface passing information in the data marshaled into the message, the

interface passing information designating to enqueue any method invocation by the first queued component on an interface of the second queued component referenced by the interface pointer reference into the second message queue." These features are discussed in the application, for example, at pages 16-17.

Applicants respectfully submit that a Wollrath-Chernick combination fails to teach or suggest the recited arrangement. For example, the Office recites the following Wollrath passages,

> Transmitting machine 401 includes a memory 404 storing objects such as objects 405 and 406, and an RMI 407 for performing processing on the objects. To transmit an object over network 400, RMI 407 uses code 409 for converting object 405 into a marshalled object that is transmitted as a byte stream 408 to machine 402. Java streams, including input and output streams, are known in the art and an explanation, which is incorporated herein by reference, appears in, for example, a text entitled "The Java Tutorial: Object-Oriented Programming for the Internet," pp. 325-53, by Mary Campione and Kathy Walrath, Addison-Wesley, 1996. *Col 6, ll 1-10.*
>
> ... A marshalled object's constructor takes a serializable object (obj) as its single argument and holds the marshalled representation of the object in a byte stream. The marshalled representation of the object preserves the semantics of objects that are at passed in RMI calls: each class in the stream is typically annotated with either the object's code or a URL to the code so that when the object is reconstructed by a call to a "get" method, the bytecodes for each class can be located and loaded, and remote objects are replaced with their proxy stubs. The "get" method is a method called by an application to execute a process of unmarshalling, which is reconstruction of an object from a marshalled object using a self-describing byte stream (the marshalled object), and a process to obtain the necessary code for that process. A proxy stub is a reference to a remote object for use in reconstructing an object. *Col. 7, ll 34-50.*

As understood by Applicants, the recited passages fail to teach or suggest a method comprising "when marshaling an interface pointer reference to the second queued component in any of the method invocations issued by the client program for the first queued component, incorporating interface passing information in the data marshaled into the message, the interface passing information designating to enqueue any method invocation by the first queued component on an interface of the second queued component referenced by the interface pointer reference into the second message queue."

Additionally, teaches away from the recited arrangement. In Chernick, when a server is remote, an RPC mechanism is used. *See e.g., col. 13, ll 10-44.* This teaches away from claim 6. For a local server, Chernick discusses a single queue shared by client and server, *see e.g., col. 13, ll*

45-67, Fig 5 and 6, which teaches away from recited queued components. Since the references when combined "must teach or suggest all the claim limitations," the proposed combination fails to teach or suggest claim 6.

For at least this reason claim 6 should be allowed. Such action is respectfully requested.

Claims 7-8

Dependent claims 7-8 recite additional patentably distinct subject matter. However, since they depend from claim 6, they are allowable for at least the reasons stated above for claim 6. Claims 7-8 should be allowable. Such action is respectfully requested.

Claim 9

Claim 9 is generally directed to a method comprising "at a later time of processing the message from the first message queue, unmarshaling the data for the method invocations from the message, re-creating the persist-able object from the object-representative data stream, issuing the set of method invocations to the queued component, and passing a reference for calling methods on the re-created persist-able object in said any method invocation to the queued component."

Specifically, claim 9 recites,

> 9. A method of yielding results from processing work of a queued component to a persist-able object, where the work of the queued component is initiated by method invocations delivered via a first message queue, the method comprising:
>
> responsive to a client program issuing a set of method invocations for the queued component, marshaling data for the method invocations of the set into a message to be enqueued into the first message queue; and
>
> when marshaling a reference for calling methods on the persist-able object in any of the method invocations issued by the client program for the queued component, persisting the persist-able object into an object-representative data stream and incorporating the object-representative data stream in the data marshaled into the message;
>
> submitting the message to the first message queue; and
>
> *at a later time of processing the message from the first message queue, unmarshaling the data for the method invocations from the message, re-creating the persist-able object from the object-representative data stream, issuing the set of method invocations to the queued component, and passing a reference for calling methods on the re-created persist-able object in said any method invocation to the queued component.* (Emphasis Added).

Applicants respectfully submit that the art cited by the Office fails to teach or suggest a method comprising "at a later time of processing the message from the first message queue, unmarshaling the data for the method invocations from the message, re-creating the persist-able object from the object-representative data stream, issuing the set of method invocations to the queued component, and passing a reference for calling methods on the re-created persist-able object in said any method invocation to the queued component." These features are discussed in the application, for example, at pages 16-17.

Applicants respectfully submit that a Wollrath-Chernick combination fails to teach or suggest the recited arrangement. For example, the Office recites the following Wollrath passages,

> Transmitting machine 401 includes a memory 404 storing objects such as objects 405 and 406, and an RMI 407 for performing processing on the objects. To transmit an object over network 400, RMI 407 uses code 409 for converting object 405 into a marshalled object that is transmitted as a byte stream 408 to machine 402. Java streams, including input and output streams, are known in the art and an explanation, which is incorporated herein by reference, appears in, for example, a text entitled "The Java Tutorial: Object-Oriented Programming for the Internet," pp. 325-53, by Mary Campione and Kathy Walrath, Addison-Wesley, 1996. *Col 6, ll 1-10.*
>
> ... When receiving machine 402 receives byte stream 408, it identifies the type of transmitted object. Machine 402 contains its own RMI 410 and code 411 for processing of objects. If byte stream 408 contains a marshalled object, machine 402 may create a new object 414 using the object type identified in the marshalled object, the state information, and code for the object. Object 414 is a copy of object 405 and is stored in memory 413 of machine 402. If code 412 is not resident or available on machine 402 and the marshalled object does not contain the code, RMI 410 uses the Java URL from the marshalled object to locate the code and transfer a copy of the code to machine 402. Because the code is Java bytecodes and is therefore portable, the receiving machine can load the code into RMI 410 to reconstruct the object. Thus, machine 402 can reconstruct an object of the appropriate type even if that kind of object has not been present on the machine before. *Col. 6, ll 30-46.*
>
> ... A marshalled object is a container for an object that allows that object to be passed as a parameter in an RMI call, but preferably postpones conversion of the marshalled object at the receiving machine until an application executing on the receiving machine requests the object via a call to the marshalled object. A container is an envelope that includes the data and either the code or a reference to the code for the object, and that holds the object for transmission. The serializable object contained in the marshalled object is typically serialized and deserialized when requested with the same semantics as parameters passed in

> RMI calls. Serialization is a process of converting an in-memory representation of an object into a corresponding self-describing byte stream. Deserialization is a process of converting a self-describing byte stream into the corresponding object. *Col. 6, ll 60-67.*
>
> ... A marshalled object's constructor takes a serializable object (obj) as its single argument and holds the marshalled representation of the object in a byte stream. The marshalled representation of the object preserves the semantics of objects that are at passed in RMI calls: each class in the stream is typically annotated with either the object's code or a URL to the code so that when the object is reconstructed by a call to a "get" method, the bytecodes for each class can be located and loaded, and remote objects are replaced with their proxy stubs. The "get" method is a method called by an application to execute a process of unmarshalling, which is reconstruction of an object from a marshalled object using a self-describing byte stream (the marshalled object), and a process to obtain the necessary code for that process. A proxy stub is a reference to a remote object for use in reconstructing an object. *Col. 7, ll 34-50.*
>
> ... The machine determines if the byte stream is a marshalled object (step 602). If it is not such an object, the machine performs normal processing of the byte stream (step 603). Otherwise, if the received byte stream represents a marshalled object, the machine holds the marshalled object for later use in response to a get method invoked by a process on the receiving machine. If the receiving machine determines that the object is to be transmitted to another machine (step 604), it simply transmits the byte stream without reconstructing the object. If the machine uses the object, it performs reconstruction of the object using its RMI and associated code (step 605). If the reconstruction code for the object is not resident on the machine, it uses a URL to request and obtain the code (step 606), as described above. The machine determines if it needs to transmit the object to another machine (step 607). If the object is destined for another machine, it is transmitted as a byte stream (step 608). *Col. 8, ll 33-50.*

As understood by Applicants, the recited passages fail to teach or suggest a method comprising "at a later time of processing the message from the first message queue, unmarshaling the data for the method invocations from the message, re-creating the persist-able object from the object-representative data stream, issuing the set of method invocations to the queued component, and passing a reference for calling methods on the re-created persist-able object in said any method invocation to the queued component."

Additionally, Chernick teaches away from the recited arrangement. In Chernick, when a server is remote, an RPC mechanism is used. *See e.g., col. 13, ll 10-44.* This teaches away from claim 9. For a local server, Chernick discusses a single queue shared by client and server, *see e.g., col. 13, ll 45-67, Fig 5 and 6,* which teaches away from recited queued components. Since the

references when combined "must teach or suggest all the claim limitations," the proposed combination fails to teach or suggest claim 9.

For at least this reason claim 9 should be allowed. Such action is respectfully requested.

Claims 10-11

Dependent claims 10-11 recite additional patentably distinct subject matter. However, since they depend from claim 9, they are allowable for at least the reasons stated above for claim 9. Claims 10-11 should be allowable. Such action is respectfully requested.

Claim 14

Claim 14 is generally directed to "a second method invocation recording component created by the queued component recorder constructor responsive to a second request of the client program to obtain a second reference for a second queued component, the second queued component having a reference passing method accepting a passed object reference as a parameter thereto, the second method invocation recording component operating in response to an invocation of the reference passing method made on the second method invocation recording component having a reference for the first method invocation recording component passed as the parameter to marshal the first method invocation recording component into a data stream representative of the invocation into a message for queuing into a message queue associated with the second queued component."

Specifically, claim 14 recites,

> 14. A computer-readable medium having thereon a computer-executable method invocation queuing system program comprising:
>
> a queued component recorder constructor operating on request of a client program to obtain a queued component reference to create a method invocation recording component and return a reference for such method invocation recording component to the client program;
>
> a first method invocation recording component created by the queued component recorder constructor responsive to a first request of a client program to obtain a first reference for a first queued component; and
>
> a second method invocation recording component created by the queued component recorder constructor responsive to a second request of the client program to obtain a second reference for a second queued component, the second queued component having a reference passing method accepting a passed object reference as a parameter thereto, the second method invocation recording component operating in response to an invocation of the reference passing method made on the second method invocation recording component having a reference for the first method invocation recording component passed as the parameter to

> marshal the first method invocation recording component into a data stream representative of the invocation into a message for queuing into a message queue associated with the second queued component.

Applicants respectfully submit that the art cited by the Office fails to teach or suggest a method comprising "a second method invocation recording component created by the queued component recorder constructor responsive to a second request of the client program to obtain a second reference for a second queued component, the second queued component having a reference passing method accepting a passed object reference as a parameter thereto, the second method invocation recording component operating in response to an invocation of the reference passing method made on the second method invocation recording component having a reference for the first method invocation recording component passed as the parameter to marshal the first method invocation recording component into a data stream representative of the invocation into a message for queuing into a message queue associated with the second queued component." These features are discussed in the application, for example, at pages 16-17.

Applicants respectfully submit that a Wollrath-Chernick combination fails to teach or suggest the recited arrangement. For example, the Office recites the following Wollrath passages,

> Transmitting machine 401 includes a memory 404 storing objects such as objects 405 and 406, and an RMI 407 for performing processing on the objects. To transmit an object over network 400, RMI 407 uses code 409 for converting object 405 into a marshalled object that is transmitted as a byte stream 408 to machine 402. Java streams, including input and output streams, are known in the art and an explanation, which is incorporated herein by reference, appears in, for example, a text entitled "The Java Tutorial: Object-Oriented Programming for the Internet," pp. 325-53, by Mary Campione and Kathy Walrath, Addison-Wesley, 1996. *Col 6, ll 1-10.*
>
> ... When receiving machine 402 receives byte stream 408, it identifies the type of transmitted object. Machine 402 contains its own RMI 410 and code 411 for processing of objects. If byte stream 408 contains a marshalled object, machine 402 may create a new object 414 using the object type identified in the marshalled object, the state information, and code for the object. Object 414 is a copy of object 405 and is stored in memory 413 of machine 402. If code 412 is not resident or available on machine 402 and the marshalled object does not contain the code, RMI 410 uses the Java URL from the marshalled object to locate the code and transfer a copy of the code to machine 402. Because the code is Java bytecodes and is therefore portable, the receiving machine can load the code into RMI 410 to reconstruct the object. Thus, machine 402 can reconstruct an object of the appropriate type even if that kind of object has not been present on the machine before. *Col. 6, ll 30-46.*

> ... A marshalled object is a container for an object that allows that object to be passed as a parameter in an RMI call, but preferably postpones conversion of the marshalled object at the receiving machine until an application executing on the receiving machine requests the object via a call to the marshalled object. A container is an envelope that includes the data and either the code or a reference to the code for the object, and that holds the object for transmission. The serializable object contained in the marshalled object is typically serialized and deserialized when requested with the same semantics as parameters passed in RMI calls. Serialization is a process of converting an in-memory representation of an object into a corresponding self-describing byte stream. Deserialization is a process of converting a self-describing byte stream into the corresponding object. *Col. 6, ll 60-67.*
>
> ... A marshalled object's constructor takes a serializable object (obj) as its single argument and holds the marshalled representation of the object in a byte stream. The marshalled representation of the object preserves the semantics of objects that are at passed in RMI calls: each class in the stream is typically annotated with either the object's code or a URL to the code so that when the object is reconstructed by a call to a "get" method, the bytecodes for each class can be located and loaded, and remote objects are replaced with their proxy stubs. The "get" method is a method called by an application to execute a process of unmarshalling, which is reconstruction of an object from a marshalled object using a self-describing byte stream (the marshalled object), and a process to obtain the necessary code for that process. A proxy stub is a reference to a remote object for use in reconstructing an object. Col. *7, ll 34-50.*
>
> ... The machine determines if the byte stream is a marshalled object (step 602). If it is not such an object, the machine performs normal processing of the byte stream (step 603). Otherwise, if the received byte stream represents a marshalled object, the machine holds the marshalled object for later use in response to a get method invoked by a process on the receiving machine. If the receiving machine determines that the object is to be transmitted to another machine (step 604), it simply transmits the byte stream without reconstructing the object. If the machine uses the object, it performs reconstruction of the object using its RMI and associated code (step 605). If the reconstruction code for the object is not resident on the machine, it uses a URL to request and obtain the code (step 606), as described above. The machine determines if it needs to transmit the object to another machine (step 607). If the object is destined for another machine, it is transmitted as a byte stream (step 608). *Col. 8, ll 33-50.*

As understood by Applicants, the recited passages fail to teach or suggest the recited arrangement.

Additionally, teaches away from the recited arrangement. In Chernick, when a server is remote, an RPC mechanism is used. *See e.g., col. 13, ll 10-44.* This teaches away from claim 14.

For a local server, Chernick discusses a single queue shared by client and server, *see e.g., col. 13, ll 45-67, Fig 5 and 6,* which teaches away from recited queued components. Since the references when combined "must teach or suggest all the claim limitations," the proposed combination fails to teach or suggest claim 14.

For at least this reason claim 14 should be allowed. Such action is respectfully requested.

Claim 15

Dependent claim 15 recites additional patentably distinct subject matter. However, since it depends from claim 14, it is allowable for at least the reasons stated above for claim 14. Claim 15 should be allowable. Such action is respectfully requested.

Claim 16

Amended claim 16 is generally directed to a computer-readable medium having recorded thereon a computer-executable method ... comprising a "queued method invocations playing component ... operating ... to invoke the method on the first queued component with a reference for the re-created method invocation recording component ... wherein the reference to the re-created method invocation recording component is passed to the first queued component so the first queued component can record a result to the second queued component."

Specifically, claim 16 recites,

> 16. A computer-readable medium having thereon a computer-executable method invocation queuing system program comprising:
>
> a queued method invocations playing component operating to retrieve a method invocations message from a message queue associated with a first queued component, the first queued component having a reference passing method accepting a passed object reference as a parameter thereto, the queued method invocations playing component further operating in response to a message containing a data stream representative of an invocation of the reference passing method having a reference for a method invocation recording component of a second queued component passed as the parameter to unmarshal the data stream, to re-create the method invocation recording component, and to invoke the method on the first queued component with a reference for the re-created method invocation recording component passed as the parameter;
>
> wherein *the reference to the re-created method invocation recording component is passed to the first queued component so the first queued component can record a result to* the *second queued component.*
>
> (Emphasis Added).

Applicants respectfully submit that the art cited by the Office fails to teach or suggest a "queued method invocations playing component ... operating ... to invoke the method on the first queued component with a reference for the re-created method invocation recording component ... wherein the reference to the re-created method invocation recording component is passed to the first queued component so the first queued component can record a result to the second queued component." These features are discussed in the application, for example, at page 16, line 13, through page 17, line 6, in view of Figures 2 and 3.

Applicants respectfully submit that a Wollrath-Chernick combination fails to teach or suggest the recited arrangement. For example, the Office recites the following Wollrath passages,

> When receiving machine 402 receives byte stream 408, it identifies the type of transmitted object. Machine 402 contains its own RMI 410 and code 411 for processing of objects. If byte stream 408 contains a marshalled object, machine 402 may create a new object 414 using the object type identified in the marshalled object, the state information, and code for the object. Object 414 is a copy of object 405 and is stored in memory 413 of machine 402. If code 412 is not resident or available on machine 402 and the marshalled object does not contain the code, RMI 410 uses the Java URL from the marshalled object to locate the code and transfer a copy of the code to machine 402. Because the code is Java bytecodes and is therefore portable, the receiving machine can load the code into RMI 410 to reconstruct the object. Thus, machine 402 can reconstruct an object of the appropriate type even if that kind of object has not been present on the machine before. *Col. 6, ll 30-46.*
>
> ... A marshalled object is a container for an object that allows that object to be passed as a parameter in an RMI call, but preferably postpones conversion of the marshalled object at the receiving machine until an application executing on the receiving machine requests the object via a call to the marshalled object. A container is an envelope that includes the data and either the code or a reference to the code for the object, and that holds the object for transmission. The serializable object contained in the marshalled object is typically serialized and deserialized when requested with the same semantics as parameters passed in RMI calls. Serialization is a process of converting an in-memory representation of an object into a corresponding self-describing byte stream. Deserialization is a process of converting a self-describing byte stream into the corresponding object. *Col. 6, ll 60-67.*
>
> ... A marshalled object's constructor takes a serializable object (obj) as its single argument and holds the marshalled representation of the object in a byte stream. The marshalled representation of the object preserves the semantics of objects that are at passed in RMI calls: each class in the stream is typically annotated with either the object's code or a URL to the code so that when the object is

> reconstructed by a call to a "get" method, the bytecodes for each class can be located and loaded, and remote objects are replaced with their proxy stubs. The "get" method is a method called by an application to execute a process of unmarshalling, which is reconstruction of an object from a marshalled object using a self-describing byte stream (the marshalled object), and a process to obtain the necessary code for that process. A proxy stub is a reference to a remote object for use in reconstructing an object. *Col. 7, ll 34-50.*

In summary, Wollrath describes a mechanism that allows an object to be passed as a parameter in an RMI call, but postpones conversion of the marshalled object at the receiving machine until an application executing on the receiving machine requests the object via a call to the marshalled object.

However, nowhere does Wollrath teach or discuss a "queued method invocations playing component ... operating ... to invoke the method on the first queued component with a reference for the re-created method invocation recording component ... wherein the reference to the re-created method invocation recording component is passed to the first queued component so the first queued component can record a result to the second queued component."

Additionally, Chernick teaches away from the recited arrangement. In Chernick, when a server is remote, an RPC mechanism is used. *See e.g., col. 13, ll 10-44.* This teaches away from claim 16. For a local server, Chernick discusses a single queue shared by client and server, *see e.g., col. 13, ll 45-67, Fig 5 and 6,* which teaches away from recited queued components. Since the references when combined "must teach or suggest all the claim limitations," the proposed combination fails to teach or suggest amended claim 16.

For at least this reason claim 16 should be allowed. Such action is respectfully requested.

Claims 17-18

New dependent claims 17 and 18 recite additional patentably distinct subject matter. However, since they depend from claim 16, they are allowable for at least the reasons stated above for claim 16. Claims 17-18 should be allowable. Such action is respectfully requested.

Claim 12

Claim 12 is generally directed to a system comprising "invoking ... a method of the server-side queued component ... comprises ... passing a reference for the client-specific queued component ... and ... invoking by the server-side queued component a method of the client-specific

queued component using the reference ... and submitting the second method invocations message to the second message queue, whereby the server-side queued component's method invocations are queued for the client-specific queued component."

Specifically, claim 12 recites,

> 12. In a distributed computing system having a multiplicity of client machines and a server machine, a method of conveying results of processing queued method invocations by a server-side queued component of a component-based server program on the server machine to a client-specific component-based program, the method comprising:
>
> associating a first message queue with the server-side queued component and a second message queue with a client-specific queued component of the client-specific component-based program;
>
> on issuing by a client program running on a first client machine in the distributed computing system a first method invocation to the server-side queued component having passed therein a reference for a client-specific queued component, recording data representative of the first method invocation into a first method invocations message, wherein said recording comprises automatically and transparently to the client program marshaling a reference to the second message queue with the data representative of the first method invocation into the first method invocations message,;
>
> submitting the first method invocations message to the first message queue;
>
> retrieving the first method invocations message from the first message queue at the server machine;
>
> unmarshaling the data representative of the first method invocation from the first method invocations message;
>
> invoking per the first method invocation a method of the server-side queued component, wherein said invoking comprises passing a reference for the client-specific queued component to the server-side queued component; and
>
> on invoking by the server-side queued component a method of the client-specific queued component using the reference passed to the server-side queued component, automatically and transparently to the server-side queued component recording data representative of the server-side queued component's method invocations using the reference passed to the server-side queued component into a second method invocations message and submitting the second method invocations message to the second message queue, whereby the server-side queued component's method invocations are queued for the client-specific queued component.

Applicants respectfully submit that the art cited by the Office fails to teach or suggest "invoking ... a method of the server-side queued component ... comprises ... passing a reference for the client-specific queued component ... and ... invoking by the server-side queued component a method of the client-specific queued component using the reference ... and submitting the second

method invocations message to the second message queue, whereby the server-side queued component's method invocations are queued for the client-specific queued component." These features are discussed in the application, for example, at pages 16-17.

Applicants respectfully submit that a Kasichainula-Chernick combination fails to teach or suggest the recited arrangement. For example, the Office recites the following Kasichainula passages,

> Upon receiving the remote call from Y' 510, Y" 511 creates a proxy Z' 512 for object Z 504 in machine 509, and creates 515 a reference table entry in which the key Z' returns a remote call reference to the proxy Z" 513 which was created by Y' 510. Then when object Y" 511 translates the call into the semantics of object Y 503 and invokes object Y, a reference to proxy object Z' 512 is passed 516 as the parameter. Thus object Y will invoke object Z' 512 when object Y's invoked method invokes the object passed in as a parameter.
>
> When object Z' is invoked 506 by object Y 503, object Z' uses the reference table entry which was created earlier 515 by object Y" 511 to determine where the call is to be directed. By looking itself up in the table 517, object Z' received a reference to object Z" 513 on machine 501. Using the received reference, object Z' translates the call into the semantics of object Z" and invokes object Z" using Remote Method Invocation or some other standard remote calling method 518.
>
> When object Z" 518 is invoked, it uses the reference table entry which was created earlier 524 by object Y' to determine where the call is to be directed. By looking itself up in the table 519, object Z" receives a reference to object Z 504. Using the received reference, object Z" translates the received call into the semantics of object Z and invokes object Z. Thus, object Y 503 has successfully invoked object Z in the course of its invocation from object X 502, even though object Y resides on a different computer than objects X and Z.
>
> Object Z 504 returns the result 521, if any, of the invocation to object Z" 513, which returns said result 522 to object Z' 512, which returns 507 said result to object Y 503. Then when object Y finishes the method which was invoked from object X 502, it returns the result 523, if any, of said invocation to object Y" 511, which returns said result to object Y' 510, which returns 508 said result to object X 502, thus completing the object X's method invocation to object Y 503, which also updated object Z 504. The proxies do all the work so that the original objects X, Y, and Z may be programmed as if they all reside on one computer. *Col. 8, ll 33-67.*

As understood by Applicants, the recited passages fail to teach or suggest "invoking ... a method of the server-side queued component ... comprises ... passing a reference for the client-specific queued component ... and ... invoking by the server-side queued component a method of the client-specific queued component using the reference ... and submitting the second method

invocations message to the second message queue, whereby the server-side queued component's method invocations are queued for the client-specific queued component."

Additionally, as understood by Applicants, Chernick fails to teach or suggest the recited arrangement. For example, the Office recites the following passages,

> The local message queue represents a very efficient and reliable transport mechanism. In one embodiment, the data may be stored in a common local memory directly available to both client and server objects. The message queue entry from the client contains the data address of the location in common memory while the returned queue receipt from the object after processing may contain the corresponding output data address. The present invention thereby avoids some of the reduction in system performance inherent with conventional network data interchange techniques by, for example, a) eliminating the need for the generation and receipt of a call receipt acknowledgement from the server to the client in response to a call; b) using a local message queue as a reliable, connected transport thereby eliminating the need to datagram package the data and handle the associated data fragment acknowledgements that would be required by a connectionless transport; and/or c) using shared local memory to eliminate the need to serialize the data during marshalling and unmarshalling for transport. *Col. 2, ll 18-29.*
>
> ...In accordance with a first embodiment of the present invention, Object Procedure Messaging, or OPM, calls are used for communications between objects. OPM calls operate in generally the same manner as conventional RPC calls for communications between objects on different platforms and in an enhanced mode for inter-object communications on a single platform. *Col. 4, ll 11-18.*
>
> ...Steps 28 and 30 are common to all intra-platform, client-server communication paths such as path 26, as well as inter-platform data paths, which utilize native RPC facilities, as will be discussed below with respect to FIG. 2. When the result of the interrogation of directory services 20 by the messaging facility of the client object determines that the server object is internal, that is locally resident, as represented by step 32, the intra-platform, inter-object communication paths as depicted in FIG. 1 diverge from the inter-platform, inter-object depicted in FIG. 2. It is important to note, however, that the next step in both intra- and inter-platform data paths uses the RPC facility. In the case of an intra-platform path, such as path 26 depicted in FIG. 1, the RPC facility is used to send the message call to message queue 22 rather than to the server object located on a separate platform.
>
> In particular, as indicated by step 34, when interrogation of directory services 20 by the messaging facility of client object 14 indicates that the called object, server object 16, is co-resident on local platform 15, the RPC facility is used to send the message. The local transport layer on local platform 15 places the message on message queue 22. The message call placed on message queue 22 includes an object handle representing server object 16. *Col. 6, ll 27-64.*

As understood by Applicants, the recited passages fails to teach or suggest "invoking ... a method of the server-side queued component ... comprises ... passing a reference for the client-specific queued component ... and ... invoking by the server-side queued component a method of the client-specific queued component using the reference ... and submitting the second method invocations message to the second message queue, whereby the server-side queued component's method invocations are queued for the client-specific queued component."

Actually, Chernick teaches away from the recited arrangement. In Chernick, when a server is remote, an RPC mechanism is used. *See e.g., col. 13, ll 10-44.* This teaches away from claim 12. For a local server, Chernick discusses a single queue shared by client and server, *see e.g., col. 13, ll 45-67, Fig 5 and 6,* which teaches away from recited queued components. Finally, Chernick requires server object awareness (*see e.g., col. 7, ll 27-30)* which teaches away from claim 12's required language—"automatically and transparently to the server-side queued component recording data representative of the server-side queued component's method invocations using the reference passed to the server-side queued component into a second method invocations message and submitting the second method invocations message to the second message queue."

Thus Chernick fails to teach or suggest "invoking ... a method of the server-side queued component ... comprises ... passing a reference for the client-specific queued component ... and ... invoking by the server-side queued component a method of the client-specific queued component using the reference ... and submitting the second method invocations message to the second message queue, whereby the server-side queued component's method invocations are queued for the client-specific queued component."

Since the references when combined "must teach or suggest all the claim limitations," the proposed combination fails to teach or suggest claim 12.

For at least this reason claim 12 should be allowed. Such action is respectfully requested.

Double Patenting

In the Office Action dated July 7, 2003, the Office rejected claims 1-5, 14-15, and 16, under the judicially created doctrine of obviousness-type double patenting, as being unpatentable over claims 5, 11, and 13 of U.S. Patent No. 6,425,017 in view of U.S. Patent No. 6,253,256. Applicants respectfully disagree.

At least some of the limitations in the claims of the present application are not taught or suggested by the claims in U.S. Patent No. 6,425,017. For example, in this application, claim 1 recites "wherein an object reference to the copy of the first method invocation recorder is passed as a parameter of the method invocation to an object of the second object class so an object of the second object class can record results for an object of the first object class," claim 14 recites "a second method invocation recording component created by the queued component recorder constructor responsive to a second request of the client program to obtain a second reference for a second queued component, the second queued component having a reference passing method accepting a passed object reference as a parameter thereto, the second method invocation recording component operating in response to an invocation of the reference passing method made on the second method invocation recording component having a reference for the first method invocation recording component passed as the parameter to marshal the first method invocation recording component into a data stream representative of the invocation into a message for queuing into a message queue associated with the second queued component," and claim 16 recites a "queued method invocations playing component ... operating ... to invoke the method on the first queued component with a reference for the re-created method invocation recording component ... wherein the reference to the re-created method invocation recording component is passed to the first queued component so the first queued component can record a result to the second queued component."

Applicant respectfully submit, that this language is not recited in the claims of U.S. Patent No. 6,425,017, nor would the claims of Patent No. 6,425,017 suggest these limitations to one of ordinary skill in the art. Additionally, no combination of U.S. Patent No. 6,425,017 with U.S. Patent No. 6,253, 256, would teach or suggest the recited language to one of ordinary skill in the art. For at least this reason, this application is allowable. Such action is respectfully requested.

Information Disclosure Statement

The Action asserts that the information disclosure statement submitted July 16, 2001, failed to include copies of all references. Applicants respectfully submit that all references were submitted with said information disclosure statements. However, Applicants herewith include a new information disclosure statement including all of said references. Additionally, the action requests copies of references discussed in the specification at page 3, lines 21-44, page 28, lines 23-26, and page 30, lines 17-19. Finally, the Action failed to consider information disclosure statements

submitted by Applicants on March 17, 2000, November 8, 2000, and July 16, 2001 (part 2). All of these references have been included in the new information disclosure statement. However, we have requested the reference discussed in the application at page 30, line 17 from the Applicants and will forward to the Examiner immediately upon receipt. Consideration of all references is respectfully requested.

Conclusion

The claims in their present form should now be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

By _____
Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446
(112607.1)